

Assignment 01

CSCI 31072 – Python Programming

Exercise 1.1

Source code:

```
1  # Exercise 1.1 - Basic Data Types
2
3  # integer
4  value = 5
5  print(f"Integer: {value}")
6  print(f"Type: {type(value)}")
7
8  # float
9  value = 5.0
10 print(f"Float: {value}")
11 print(f"Type: {type(value)}")
12
13 # string
14 value = "hello world"
15 print(f"String: {value}")
16 print(f"Type: {type(value)}")
17
18 # boolean
19 value = True
20 print(f"Boolean: {value}")
21 print(f"Type: {type(value)}")
```

Output:

```
● $ python main.py
○ Integer: 5
  Type: <class 'int'>
  Float: 5.0
  Type: <class 'float'>
  String: hello world
  Type: <class 'str'>
  Boolean: True
  Type: <class 'bool'>
```

Using Type operator print the type of the variable.

Exercise 1.2

source code:

```
# Exercise 1.2 - type conversion between data types

# converting integer to float
integer_value = 5
float_value = float(integer_value)
print(f"Type of converted value: {type(float_value)}")

# converting float to integer
float_value = 5.0
integer_value = int(float_value)
print(f"Type of converted value: {type(integer_value)}")

# converting integer to string
integer_value = 5
string_value = str(integer_value)
print(f"Type of converted value: {type(string_value)}")

# converting string to integer
string_value = "5"
integer_value = int(string_value)
print(f"Type of converted value: {type(integer_value)}")

# converting string to float
string_value = "5.0"
float_value = float(string_value)
print(f"Type of converted value: {type(float_value)}")
```

```

# converting float to string
float_value = 5.0
string_value = str(float_value)
print(f"Type of converted value: {type(string_value)}")

# converting boolean to string
boolean_value = True
string_value = str(boolean_value)
print(f"Type of converted value: {type(string_value)}")

# converting string to boolean
string_value = "True"
boolean_value = bool(string_value)
print(f"Type of converted value: {type(boolean_value)}")

# converting integer to boolean
integer_value = 5
boolean_value = bool(integer_value)
print(f"Type of converted value: {type(boolean_value)}")

# converting boolean to integer
boolean_value = True
integer_value = int(boolean_value)
print(f"Type of converted value: {type(integer_value)}")

# converting float to boolean
float_value = 5.0
boolean_value = bool(float_value)
print(f"Type of converted value: {type(boolean_value)}")

```

Output:

```

$ python main.py
Type of converted value: <class 'float'>
Type of converted value: <class 'int'>
Type of converted value: <class 'str'>
Type of converted value: <class 'int'>
Type of converted value: <class 'float'>
Type of converted value: <class 'str'>
Type of converted value: <class 'str'>
Type of converted value: <class 'bool'>
Type of converted value: <class 'bool'>
Type of converted value: <class 'int'>
Type of converted value: <class 'bool'>
Type of converted value: <class 'float'>

```

Using type casting converting one data type to supported another data type.

Exercise 2.1

Source code:

```
# Exercise 2.1: String Manipulation

# Concatenation
string1 = "Hello"
string2 = "World"
concatenated_string = string1 + " " + string2
print(f"Concatenated string: {concatenated_string}")

# Slicing
string = "Hello World"
sliced_string = string[0:5]
print(f"Sliced string: {sliced_string}")

# Formatting
name = "Pawan"
age = 23
formatted_string = f"My name is {name} and I am {age} years old."
print(f"Formatted string: {formatted_string}")
```

Output:

```
● $ python main.py
Concatenated string: Hello World
Sliced string: Hello
Formatted string: My name is Pawan and I am 23 years old.
```

Demonstrate some of the main operations in string. String Concatenation can be done using a simple + operator. Slicing itself denotes slicing used to slice the string into two parts. Using the short method f'' we can format the string in Python.

Exercise 2.2

Source code:

```
# Exercise 2.2: Palindrome Checker

text = "madam"

def palindrome_checker(text):
    reversed_text = text[::-1]
    if text == reversed_text:
        print(f"{text} is a palindrome")
    else:
        print(f"{text} is not a palindrome")

palindrome_checker(text)
```

Output:

```
● $ python main.py
  madam is a palindrome
```

The function `palindrome_checker` function first reversed the given text. Check the original text and the reversed version. If both are the same that's a palindrome.

Exercise 3.1

Source code:

```
# Exercise 3.1: Simple Calculator

def calculator(num1, num2, operator):
    if operator == "+":
        return num1 + num2

    elif operator == "-":
        return num1 - num2

    elif operator == "*":
        return num1 * num2

    elif operator == "/":
        return num1 / num2

    else:
        return "Operator not valid"

# add
num1 = 5
num2 = 3
operator = "+"
result = calculator(num1, num2, operator)
print(f"Addition: {result}")

# subtract
num1 = 5
num2 = 3
operator = "-"
result = calculator(num1, num2, operator)
print(f"Subtraction: {result}")
```

Output:

```
$ python main.py
Addition: 8
Subtraction: 2
Multiplication: 15
Division: 1.6666666666666667
Invalid operator: Operator not valid
```

The calculator function accepts 3 arguments. Two operands and a single operator. Based on the passing value for operator action will change.

Exercise 3.2

Source code:

```
# Exercise 3.2: Grading System

def grading_system_marks(marks):
    if marks >= 75:
        return "A"
    elif marks >= 65:
        return "B"
    elif marks >= 55:
        return "C"
    elif marks >= 35:
        return "S"
    else:
        return "F"

# marks
marks = 85
grade = grading_system_marks(marks)
print(f"Grade: {grade}")

marks = 68
grade = grading_system_marks(marks)
print(f"Grade: {grade}")

marks = 59
grade = grading_system_marks(marks)
print(f"Grade: {grade}")

marks = 40
grade = grading_system_marks(marks)
```

Output:

```
● $ python main.py
○ Grade: A
  Grade: B
  Grade: C
  Grade: S
  Grade: F
```

The `grading_system_marks` function accepts marks as arguments and based on the range of those marks returns relevant grades as a string.

Exercises 4.1 and 4.2

Source code:

```
# Exercise 4.1: Sum of Numbers

sum_from_1_to_10 = sum(range(1, 11))
print(f"Sum from 1 to 10: {sum_from_1_to_10}")

# Exercise 4.2: Fibonacci Sequence

def fibonacci_sequence(n = 10):
    start, next = 0, 1

    while n > 0:
        print(start)
        start, next = next, start + next
        n -= 1

fibonacci_sequence(10)
```

Output 4.1:

```
$ python main.py
Sum from 1 to 10: 55
```

Range function create a list of numbers 1 to 10 and sum function will add all those numbers and return the summation.

Output 4.2:

```
$ python main.py
0
1
1
2
3
5
8
13
21
34
```

In Fibonacci function while loop run until $n > 0$ times and return the Fibonacci series of numbers.

Exercise 5.1

Source code:

```
# Exercise 5.1: Factorial Function

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

n = 5
value = factorial(n)
print(f"Factorial of {n}: {value}")
```

Output:

```
• $ python main.py
  Factorial of 5: 120
```

This is a nested function calling. The factorial function is calling recursively until given $n == 0$ condition is satisfying. Then calculate the factorial value and print it.

Exercise 5.2

Source code:

```
# Exercise 5.2: Maximum of Three Numbers

def maximum_of_three_numbers(num1, num2, num3):
    if num1 > num2 and num1 > num3:
        return num1
    elif num2 > num1 and num2 > num3:
        return num2
    else:
        return num3

num1, num2, num3 = 5, 10, 3
maximum = maximum_of_three_numbers(num1, num2, num3)
print(f"Maximum of {num1}, {num2}, {num3}: {maximum}")
```

Output:

```
• $ python main.py
  Maximum of 5, 10, 3: 10
```

This function accepts 3 arguments and in each condition, it checks which number is maximum. If any condition is satisfied it will return the maximum value among the given inputs.

Exercises 6.1 and 6.2

Source code:

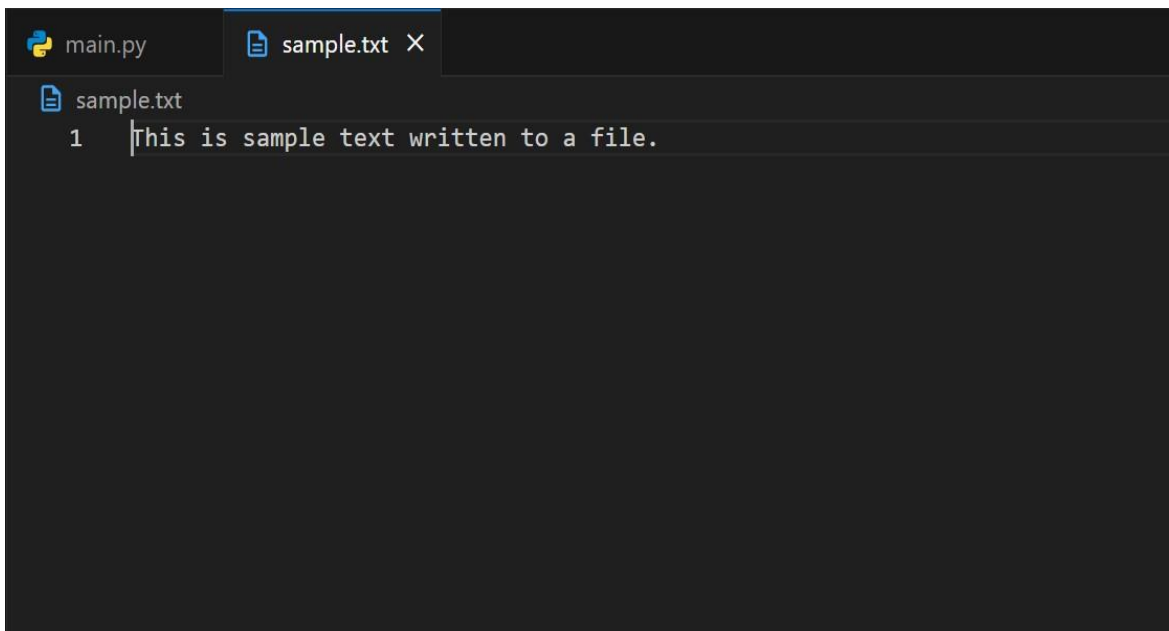
```
# Exercise 6.1: Writing to a File

with open("sample.txt", "w") as f:
    f.write("This is sample text written to a file.")

# Exercise 6.2: Reading from a File

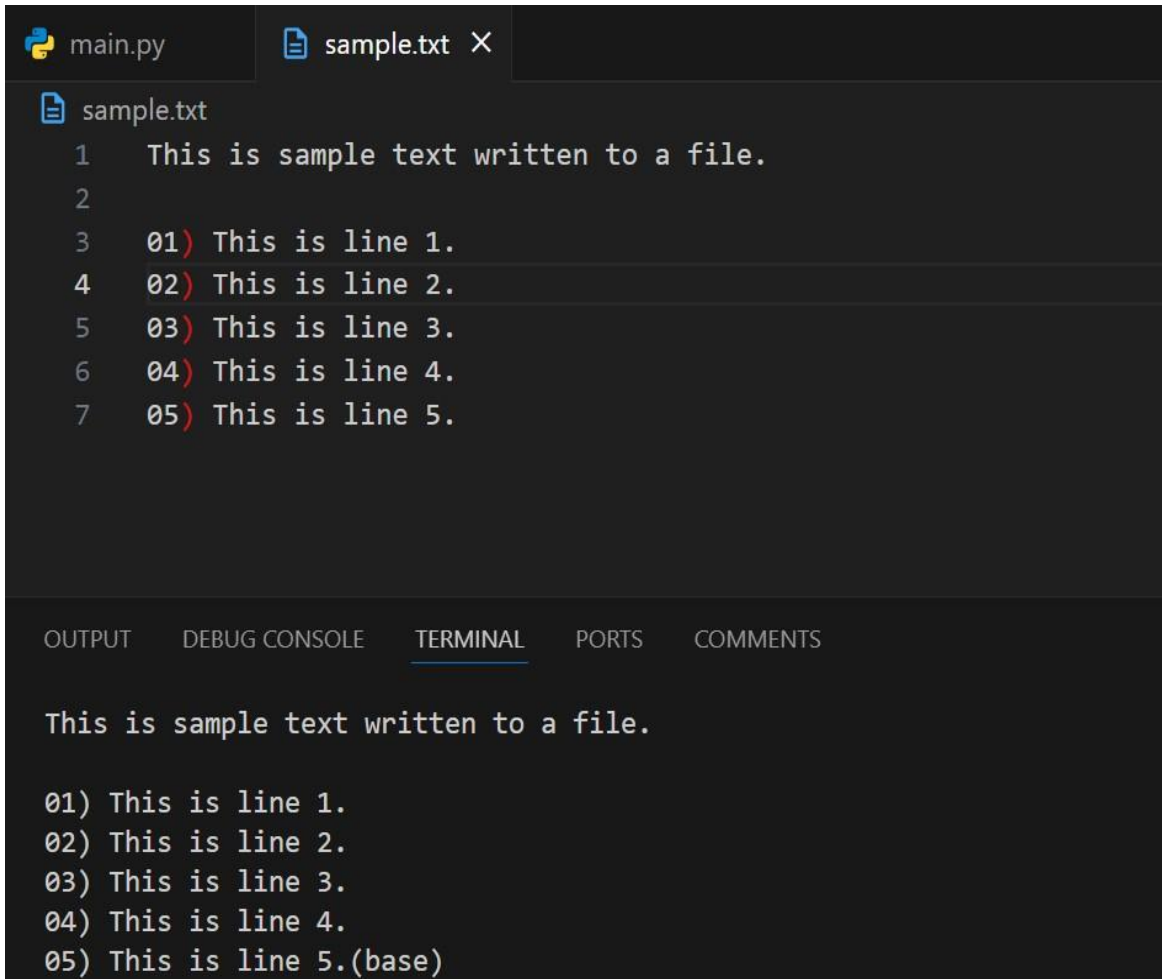
with open("sample.txt", "r") as f:
    content = f.read()
    for line in content:
        print(line, end="")
```

Output 6.1:



```
main.py  sample.txt X
sample.txt
1  This is sample text written to a file.
```

Output 6.2:



The screenshot shows a Jupyter Notebook interface with two tabs at the top: 'main.py' and 'sample.txt'. The 'sample.txt' tab is active, displaying the following text:

```
1 This is sample text written to a file.  
2  
3 01) This is line 1.  
4 02) This is line 2.  
5 03) This is line 3.  
6 04) This is line 4.  
7 05) This is line 5.
```

Below the code editor, there is a terminal output section with tabs for 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is selected, showing the following output:

```
This is sample text written to a file.  
  
01) This is line 1.  
02) This is line 2.  
03) This is line 3.  
04) This is line 4.  
05) This is line 5.(base)
```

For both exercises first need to open the file and load it into memory. That is what I have done using with open() function. The second argument to open() function denotes whether we need to read the file or write to the file.

when file reading since there are a couple of lines I ran a for loop and printed those lines separately.

Google Collab Link for source code:

<https://colab.research.google.com/drive/1HmdzK0D3TzzVCm6ttOz8V8rwPeg4l4Mk?usp=sharing>